Automated (Re-)Design of Software Agents

Brazier, F.M.; Wijngaards, N.J.E.

published in
Proceedings of the Artificial Intelligence in Design Conference (AID2002)
2002

document version
Publisher's PDF, also known as Version of record

AUTOMATED (RE-)DESIGN OF SOFTWARE AGENTS

FRANCES M.T. BRAZIER AND NIEK J.E. WIJNGAARDS
Intelligent Interactive Distributed Systems Group, Division of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands
Email: {frances, niek}@cs.vu.nl URL: http://www.iids.org

Abstract. Autonomous software agents are dynamic entitities: they are capable of discovering a need for change - for additional knowledge and/or functionality on the basis of their analysis of specific situations. Agent factories are capable of redesigning and reactivating agents on the basis of the information provided by agents and/or knowledge available within the agent factories. As a result agents may evolve in ways their designers could never have pre-conceived. The artefacts themselves are dynamic in a way that can not be compared to any other type of design in current design practice. A number of prototype agents and agent factories have been built to evaluate the feasibility of this concept and its consequences.

1. Introduction

Design and re-design are often intertwined. In a changing world in which needs, desires, requirements, manufacturing processes and technology continually change, designs often need to adapt. New designs are often based on existing designs: the question as to whether a design process is a re-design process or not, is not always easily answered or relevant. Human designers identify new needs and desires, and designs evolve.

In some cases of automated design processes, however, the situation may slightly differ. Automated software design is an example of a domain in which reference models, components and rules are used to automatically design an artefact (software). In many cases the design process is deterministic and can be fully tracked. It becomes more interesting when the software designed has a will of its own. This is the case for software agents. Intelligent software agents are autonomous pieces of code: they typically act in unpredictable environments on the basis of changing

knowledge. They are designed to learn from their interaction with their environment and communication with other agents. They can also be designed to discover their own limitations. They have a degree of self-awareness and evolve.

This paper describes an approach to automated re-design of agents, based on agents' self-awareness. Agents are capable of discovering a need for additional knowledge and/or functionality. On the basis of their analysis of specific situations and their mandates (the need to act in a given situation) agents can decide to be adapted. Agent factories provide this functionality: agent factories are capable of adapting agents' code and state and re-activating them. Agent factories perform the re-design. The agents themselves decide which agent factory they trust to perform a re-design task[°]. In contrast to non-automated design the artefact has acquired an active role in the design process. As a result an agent may evolve in a way its designer could never have pre-conceived: the design artefact becomes a dynamic artefact.

This paper describes the principles behind adaptive artefacts of the kind described above. Section 2 addresses the phenomenon of self-awareness in agents, Section 3 sketches the re-design process and Section 4 illustrates this process for an information retrieval agent. Section 5 discusses the results and places the results in the context of current and future research.

2. Agents' Self-Awareness

This section focuses on the role/meaning of self-awareness of agents. Section 2.1 introduces the concept of an adaptive system. Section 2.2 describes the types of knowledge, information and functionality needed to acquire self-awareness. Section 2.3 discusses degrees of self-awareness.

2.1 ADAPTIVE SYSTEMS

Adaptive systems, or dynamic artefacts, are artefacts that change due to changes in their environment. Examples of dynamic artefacts include buildings that adjust lighting and temperature on the basis of occupation of rooms (Mozer, 1999), elevators that second-guess the behaviour of their clientele, auto-pilots of aeroplanes, which take, and relinquish, control to the human pilots, or self-configuration of autonomous (spacecraft) systems (Williams and Nayak, 1996).

[°] In theory the agent factory could be part of each and every agent but in current prototypes these functions have been separated.

Software agents are a specific type of autonomous systems that adapt as a result of interaction with their environment and/or communication with other agents. Personification is an example: information gathering agents often maintain profiles of other agents (possibly human (Wells and Wolfers, 2000)), and adapt these profiles on the basis of interaction with these agents.

Another example of adaptive agent behaviour, e.g. by (Rus, Gray and Kotz, 1996) in distributed information gathering, occurs when an agent is capable of abandoning a previous goal or plan, and adopting a new goal or plan which better fits the current situation of the agent. Most often the learning techniques involved determine the type and level of adaptation e.g. as described by (Reffat and Gero, 2000; Grefenstette, 1992).

2.2 SELF-AWARENESS: TYPES OF KNOWLEDGE, INFORMATION AND FUNCTIONALITY

Self-aware agents need to have the following knowledge, information, and functionality:

- Self-awareness knowledge: the agent must have knowledge that describes the agent's functionality and behaviour, as well as non-functional characteristics of the agent.
- Monitoring information: the agent must be able to monitor its functioning and behaviour in a given situation.(e.g. when it wishes to be adapted and re-activated),
- Self-assessment knowledge: the agent must have knowledge that determines the extent to which an agent may function in a given situation
- Need formulation knowledge: the agent must have knowledge with which needs for adaptation can be determined
- Integration: self-awareness knowledge, monitoring information, self-assessment knowledge, and need formulation knowledge must be integrated in the (internal) functioning of a self-aware agent.
- Communication: the agent must have knowledge of the way in which it can choose an agent factory, and interact with an agent factory (e.g. protocols and languages for either an intermediary or direct interaction with the factory).

How this knowledge, information and functionality is acquired depends on an agent’s design. If, for example, an agent’s architecture is based on a generic agent model (e.g. Brazier, Jonker and Treur, 2000) with a specific component for internal "own process control", this component may be a perfect candidate for self-awareness knowledge and functionality.

2.3 DEGREE OF SELF-AWARENESS

An adaptive agent needs to be aware of its own abilities (which may also be named skills, or tasks, or behaviours, etc.). The knowledge about its abilities may be simple in form (e.g., facts), or more elaborate (e.g., a model of its own behaviour). Most important is that an adaptive agent is able to recognise, or predict, situations in which it may, or may not, function correctly.

Self-awareness knowledge is imprecise by nature. Although the absence, or presence, of certain abilities may result in a strong belief in success or failure, only by actually performing a certain task, can an adaptive agent discover actual success.

Irrespective of the form of the knowledge, an agent has a certain *degree of self-awareness*. At the one extreme of the spectrum is total non-awareness: the agent does not know which factors determine success or failure of a task. At the other extreme of the spectrum is total self-awareness: the agent fully understands its own abilities, and is able to predict with a high degree of certainty whether or not it is able to perform a certain task.

The less precise an agent's self-awareness knowledge, the less precise its needs for adaptation can be formulated, and the more responsibility is placed on an agent factory.

3. Adapting Agents

An agent factory is a facility that creates, and modifies, software agents. Section 3.1 describes the principles on which the agent factory is based. Section 3.2 describes the use of building blocks in the agent factory.

3.1 AGENT FACTORY

An agent factory designs and adapts agents. The agent factory is based on five underlying assumptions (Brazier and Wijngaards, 2001a): (1) agents have a compositional structure, (2) re-usable parts of agents can be identified, (3) two levels of descriptions are used: conceptual and detailed, (4) properties and knowledge of properties are available, and (5) no commitments are made to specific (programming or modelling) languages and/or ontologies.

The design of an agent within the agent factory is based on configuration of building blocks: a blueprint. Building blocks include cases and partial (agent) designs (cf. generic models / design patterns). This approach is related to design patterns (e.g., Gamma, Helm, Johnson and Vlissides, 1994;

Peña-Mora and Vadhavkar, 1996), libraries of software with specific functionality (e.g., problem-solving models (Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, van de Velde, and Wielinga, 1999), and generic task models (Chandrasekaran, 1986; Brazier, Jonker, and Treur, 2000)).

The configuration-based approach relates to, e.g., model-based re-configuration (Stumptner and Wotawa, 1998), design using re-usable design patterns (Peña-Mora and Vadhavkar, 1996), and IBROW (Motta, Fensel, Gaspari and Benjamins, 1999), in which re-usable parts of problem-solving methods are 'configured'.

An agent factory includes a number of processes, some of which are involved with account management (which client requires which quality of service), agent packaging and handling (how to prepare an agent for transfer to an agent platform), etc. One of the processes, the design centre, is responsible for the (re-)design of agents. The design centre is based on a model for (re-)design of compositional systems (Brazier, Jonker, Treur and Wijngaards, 2001), in which explicit reasoning about strategies, requirements, and artefact descriptions is modelled. The blueprint of an agent functions as both its functional specification, but also as (part of the) design rationale concerning the agent (Peña-Mora and Vadhvakar, 1996).

Within the design centre, needs for adaptation are interpreted into disambiguated, apparently non-conflicting, and specific requirements on the basis of which the blueprint of an agent may be modified. Strategies play an important role in this process: not only are strategies needed to decide when to manipulate requirements, and then to manipulate blueprints, but strategic knowledge is also needed to guide the re-design process: how to (re)solve conflicting requirements, what to modify in a blueprint to satisfy given requirements, etc.

The needs for adaptation, specified by an adaptive agent, are translated into sets of qualified requirements, on the basis of which the blueprint of the agent is modified. The process of modifying an agent's blueprint entails manipulating the configuration of conceptual building blocks, the configuration of detailed building blocks, and the mapping between these configurations. The library of building blocks is limited in size, and building blocks with specific characteristics (functional and non-functional) need to be retrieved, to be (correctly) configured with other building blocks. Partially matching building blocks may need to be adapted (by other building blocks) before combined with configured building blocks, etc.

The co-ordination of the process of manipulation of sets of qualified requirements and the manipulation of blueprints, is not trivial. When to switch manipulation processes, how to synchronise parallel activities, how to allocate time and resources to each manipulation process, are of importance.

Strategic knowledge is needed to decide when to issue which design strategies to one, or both, manipulation processes. And within each manipulation process, strategic knowledge is needed to decide what to work on, and when to do what. For example, multiple interpretations may exist for a need for adaptation, each interpretation alternative may be in a different 'context'. Strategic knowledge is needed to determine which context to choose.

3.2 BUILDING BLOCKS

In the agent factory building blocks are either components with open slots, fully specified components, and/or a combination of both. Building blocks are defined at one of the two levels of detail: conceptual and detailed. As a result, a mapping is needed between building blocks at conceptual level and building blocks at detailed level. (Note that this mapping may be structure preserving, but that this is not necessarily ideal.) A detailed description of a building block includes the operational code For each conceptual description, a number of detailed descriptions may be devised and vice versa. These detailed descriptions may differ in the operational language (e.g., C, C++, Java), but also in, for example, the efficiency of the operational code. The conceptual descriptions may differ in the modelling paradigm (e.g., UML, DESIRE), but also in, e.g., the detail in which an agent's functionality is modelled. In the current prototypes of the agent factory building block descriptions in DESIRE, UML, C, C++ and Java are supported.

Building blocks themselves are configurable, but cannot be combined indiscriminately. The *open slot* concept is used to regulate the ways in which components may be combined. An open slot in a component has associated properties at both levels of detail that prescribe the properties of the entity to be 'inserted'.

Specific 'glue' may be needed to aid the insertion of a building block in an open slot. Glue, which exists at both conceptual and detailed levels of design, is used to transform certain information to the correct format/ontology.

A mapping between building blocks relates a building block containing a conceptual description to a building block containing a detailed description. The mapping relates open slots of the conceptual building block to the open slots in the detailed building block.

4. Example

The examples in this section focus on a self-aware information retrieval

agent and an agent factory[*]. A scenario is described in which a self-aware information retrieval agent discovers a need to adapt, finds and enters an agent factory and is adapted according to its needs. Afterwards, the agent continues its task. The elements in the scenario are shown in Figure 1. In this example, it is assumed that information is annotated by an ontology framework designed for the Semantic Web.

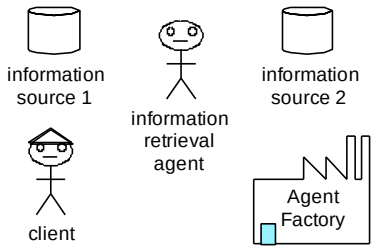

Figure 1. The actors in the scenario: a client of the information retrieval agent, the information retrieval agent, an agent factory, and a number of information sources.

Section 4.1 describes the scenario from the point of view of the adaptive information retrieval agent. The re-design process itself, is performed by an agent factory. Section 4.2 describes the re-design part of the scenario, from the point of view of the agent factory. Section 4.3 describes other examples.

4.1 EXAMPLE OF AGENT SELF-AWARENESS

A client, possibly a human agent, issues a request for an airline reservation to an information retrieval agent. The information retrieval agent consults the specified information source, and discovers it cannot interpret information in that information source. Suppose, for example, that the information source is expressed in a Dutch vocabulary for seat-reservation of aeroplanes, and the agent only understands an English version of the vocabulary.

The information retrieval agent has a certain degree of self-awareness. The extent, or granularity, of its self-awareness influences the ability of the agent to formulate its needs for adaptation.

In our example the information retrieval agent possesses the following facts concerning interaction with other agents and external objects:

```
I_speak_protocol( http )
I_speak_protocol( fipa_acl_v1.0b )
I_understand_ontology_framework( xml )
```

[*] Issues such as security, authorisation, and payment (Wayner, 1995) are not discussed in this paper.

I_understand_ontology_framework(rdf)

I_understand_ontology_framework(daml+oil)

I_understand_ontology(airplane_seat_reservations_English)

The information source, e.g., annotated in DAML+OIL (Horrocks, van Harmelen, Patel-Schneider, Berners-Lee, Brickley, Connoly, Dean, Decker, Fensel, Hayes, Heflin, Hendler, Lassila, McGuinness and Stein, 2001) has a meta-description which states the following characteristics of its contents:

expressed_in_ontology_framework(daml+oil)

expressed_in_ontology(airplaine_seat_reservations_Dutch)

The information retrieval agent is able to determine, on the basis of this information, that it is unable to understand the information source. Although it may download the specific ontology (e.g., by using Sesame by Broekstra, Kampman and Van Harmelen, 2001), it doesn't know how to use the ontology, as it doesn't have any understanding of the ontology. The agent formulates a request for adaptation to the agent factory, in which the following is expressed:

request(possibility_for_adaptation)

required_functionality(I_understand_ontology(
airplane_seat_reservations_Dutch))

The agent factory acknowledges the request and the information retrieval agent contacts a directory service, discovers an agent factory close by that it trusts and is capable of performing the desired change, and migrates to this agent factory, where it is subsequently adapted. The information retrieval agent is, in this example, not 'alive' during its adaptation. When the agent factory has finished adapting the information retrieval agent, the agent is sent back to its original location, and 'awakened'.

The information retrieval agent becomes awake, receives a message from the agent factory with details about the success of its adaptation, and adds a new fact to its self-awareness knowledge:

I_understand_ontology(airplane_seat_reservation_Dutch)

The information retrieval agent continues its original task, and consults the information source in order to satisfy the request sent by its client.

4.2 AGENT FACTORY EXAMPLE

Agents with a high degree of self-awareness are able to formulate specific needs for adaptation. For example, the information retrieval agent issued the following requirements for adaptation:

request(possibility_for_adaptation)

required_functionality(I_understand_ontology(
airplane_seat_reservations_Dutch))

For an agent to be adapted, yet retain its 'memory' after adaptation, an

important issue needs to be resolved. This involves not only sending its blueprint to the agent factory, but also sending its *memory* (because its memory may need to be changed to 'fit' its adapted blueprint).

It is assumed that the blueprint of the information retrieval agent has become available to the agent factory. On the basis of the blueprint of the information retrieval agent, and the desired adaptation, the agent factory adapts the blueprint of the agent. This involves a number of steps.

One of the first steps is refining the needs for adaptation into more specific requirements. E.g., the need for adaptation

```
required_functionality( I_understand_ontology(
                        airplane_seat_reservations_Dutch ) )
```

may be refined into a qualified requirement as shown below:

```
requirement( hard, interpret ontology(
                    airplane_seat_reservations_Dutch )
                as ontology(
                    airplane_seat_reservations_English ) )
```

That is, as both ontologies are quite similar, a mapping may be devised by which the Dutch ontology is interpreted in terms of the English ontology. This requirement may be further refined, resulting in a set of qualified requirements which can be (temporarily) committed to by the agent factory.

The blueprint of the agent is modified in accordance with this more specific set of qualified requirements. Figure 2 shows part of the conceptual building block configuration of the agent before it went to the agent factory, by only focussing on functional aspects (not including information flow within the agent). The abbreviation "BB/kb" denotes a conceptual building block containing a knowledge-base; "BB/comp" denotes a conceptual building block containing a component, which may be composed of other components.

The agent is based on a generic agent model (Brazier, Jonker and Treur, 2000), in which separate processes are distinguished for control of the agent (own process control), interaction with objects in the external world (world interaction management), interaction with agents (agent interaction management), and specific tasks of an agent (agent specific task). In addition processes for maintenance of information on the world, or other agents are distinguished (not present in this specific agent).

The adaptation of an agent is a re-design process. The re-design process within the Agent Factory is based on the generic model of design (Brazier, Langen, Ruttkay and Treur, 1994). This model explicitly distinguishes between reasoning about the overall co-ordination of the (re-)design process, manipulation of sets of qualified requirements, and manipulation of blueprints.

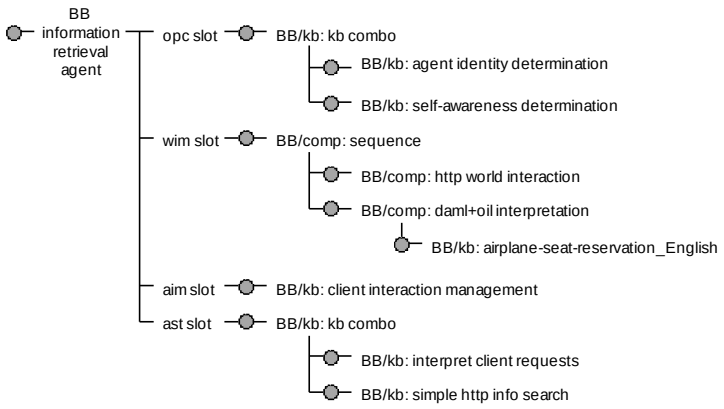

Figure 2. Part of the initial building block configuration of the information retrieval agent.

Although an agent is configured from building blocks, many alternatives may be considered during the re-design of an agent's blueprint. Strategic knowledge is used to guide the re-design process (Brazier, Langen and Treur, 1998; Brazier, Splunter and Wijngaards, 2001).

As an example of a situation in which strategic knowledge is needed, consider the following. Although the original version of the agent understands the English ontology for aeroplane seat reservations, this understanding is based on an old version of that ontology. For both the English and Dutch ontologies, new versions are available. The issue to be resolved is whether the old version of the Dutch ontology is used, or the English version is updated first, and then the new version of the Dutch ontology is used. The agent has not specified any need for adaptation concerning an update of its ontologies. As the agent is being re-designed, it is not available for comments, so it cannot aid in resolving the issue. The agent factory's strategic knowledge is responsible for resolving this issue.

The re-design process iterates between manipulating sets of qualified requirements, and manipulating blueprints (in order to satisfy requirements from a temporarily committed-to set of qualified requirements). After a number of these iterations, a blueprint has been devised which satisfies a set of qualified requirements, which is an interpretation of the needs for adaptation. Figure 3 shows the resulting conceptual building block configuration after adaptation of the agent by the agent factory.

The blueprint of an agent consists of both a configuration of conceptual building blocks and a configuration of detailed building blocks. Strategic knowledge is required to decide when to focus on which of the two configurations (either conceptual or detailed) of building blocks. In this example, the decision is to first complete the configuration of conceptual

building blocks before modifying the configuration of detailed building blocks.

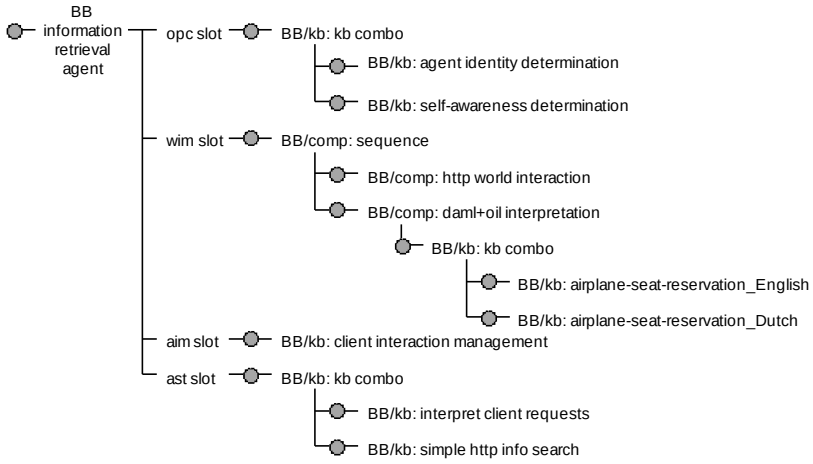

Figure 3. Part of the resulting building block configuration of the information retrieval agent.

The rationale for this decision being that modifications in the configuration of detailed building blocks are more difficult to accomplish (and revoke), than modifications to the configuration of conceptual building blocks. With respect to the relation between conceptual and detailed building blocks another strategy is employed:

> New conceptual building blocks, need to be related to detailed building blocks for which a number of specific non-functional properties are the *same* (in this case: programming language and slot format) as in the detailed building blocks in the current configuration.

In our case this ensures that the agent is built entirely from detailed building blocks written in, e.g., Java and adhering to a specific format for the slots of the detailed building blocks. This strategy should most likely be replaced by the strategy that:

> New conceptual building blocks, need to be related to detailed building blocks with non-functional properties that can co-exist with the non-functional properties (in this case: programming language and slot format) of the detailed building blocks in the current configuration.

The latter, alternative strategy would make it possible for, e.g., detailed building blocks written in Prolog to be 'wrapped' by a Java-based Prolog interpreter, and used in conjunction with other Java-based detailed building blocks.

A problem that may occur, is when none of the related detailed building blocks can be combined with the detailed building blocks in the current configuration. Again strategic knowledge is needed to decide what to do,

e.g., should alternatives be sought for detailed building blocks in the current configuration, or should the configuration of conceptual building blocks be modified? In the current example the last option was chosen.

The assembly process is a subprocess of the agent factory, which assembles an executable agent on the basis of the configuration of detailed building blocks and compiles this into executable code. In the current prototype, this is a fairly straightforward process. However, during assembly incompatibilities between detailed building blocks may be discovered. Such incompatibilities are resolved by reactivating the re-design process. The binary, executable version of the agent, plus the memory of the information retrieval agent, is used to 'awaken' the agent in its original location with its new functionality.

4.3 OTHER EXAMPLES

A self-aware agent may have knowledge about its own self-awareness: its degree and extent of self-awareness with respect to different situations. This enables an agent to formulate a need for adaptation concerning precisely this aspect. For example, if an agent notices it inability to succesfully perform a number of tasks, it may formulate a need for adaptation to this purpose.

A self-aware agent may also employ an agent factory to (temporarily) remove part of its functionality, e.g. because it is too large (in binary size, an important aspect for mobile self-aware agents), or too computationally expensive. At a later point in time, it may request to have the functionality re-installed.

5. Discussion and Future Research

The agent factory is compared to a number of related approaches in Section 5.1, after which our research on agent factories is compared to our previous research. Results of this paper are discussed in Section 5.3.

5.1 COMPARISON

The agent factory can be compared to component-based development, agent construction kits, software reusability, case-based reasoning, and configuration design.

The agent factory's approach to combining components seems similar to the approach taken in *component-based development (CBD)* of software (Sparling, 2000). One distinction with our approach is that our approach includes annotations of components at two levels of abstraction (conceptual and operational). In CBD, interfaces are described for components (which

are independent of an operational language); this correlates to the open slots in components. From our perspective CBD provides a useful means to describe operational descriptions of the building blocks used by the agent factory.

Currently a relatively large number of tools and/or frameworks exists for the (usually semi-automatic) *creation of agents*, however not automated adaptation. Examples include e.g. AgentBuilder (Reticular Systems, 1999), D'agents/AgentTCL (Gray, Kotz, Cybenko, and Rus, 1997), ZEUS (Nwana, Ndumu, Lyndon, and Collis, 1999), NOMADS (Suri, Bradshaw, Breedy, Groth, Hill, Jeffers, Mitrovich, Pouliot and Smith, 2000), Sensible Agents (Barber, McKay, MacMahon, Martin, Lam, Goel, Han and Kim, 2001), and Tryllian's Agent Development Kit (Tryllian, 2001). All of these approaches commit to a specific operational description of agents, and usually also commit to a specific conceptual description of their agents. The agent factory does not make such commitments, which makes the agent factory more general purpose (with all the common advantages and disadvantages).

The agent factory aims at pragmatically circumventing a number of issues related to *software reusability* (e.g., Biggerstaff and Perlis, 1997). A major problem is annotating reusable pieces of software such that they can be retrieved at a later time (by other people) and reused with a minimal number of changes (Peña-Mora and Vadhvakar, 1996). In the agent factory the latter is endeavoured as well. The former is currently solved in a pragmatic way: components are annotated, and, when needed, a mapping is provided to other annotations. This, however, is not a scalable solution, and, as such, one of our current foci of research. An important decision concerning standardisation is that the agent factory does not aim to adhere to one specific standard, but a number of standards.

In *case-based reasoning* approaches (e.g., Kolodner, 1993; Maher and Pu, 1997) libraries of cases are consulted to find a case which matches a problem, upon which the retrieved case is adapted. This approach differs from the agent factory in that cases are modified internally, instead of combined with other cases. Techniques for retrieving cases from case libraries are, of course, relevant to retrieving components from libraries.

The approaches taken by *design-as-configuration* (e.g., as described in (Stefik, 1995), CommonKads (Schreiber, Akkermans, Anjewierden, de Hoog, Shadbolt, van de Velde, and Wielinga, 1999), and elevator configuration (Schreiber and Birmingham, 1996)) focus on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). In most of these approaches no explicit manipulation of requirements is present, nor is a multi-levelled description of the elements taken into account. Models and theories on configuration-based design are relevant to the agent factory, in particular to the processes involved in combining conceptual and operational

descriptions.

5.2 COMPARISON TO PREVIOUS RESEARCH

Previous research (Brazier, Jonker and Treur, 2000; Brazier, Jonker, Treur and Wijngaards, 2000 & 2001) focussed on automated redesign of multi-agent systems at a conceptual level. The current research extends this work in three aspects.

The first distinction with previous work is that the agent factory no longer focuses on re-designing agents on the basis of first principles at a conceptual level. The agent factory uses building blocks to construct, and adapt, agents. Building blocks are reusable parts of agents, ranging from skeletons for larger parts of agents (i.e., templates) to specific functionality (i.e., components).

The second distinction with previous work is a broadening of the scope of the re-design process. The agent factory modifies not only the conceptual description of an agent, but also its detailed description (operational code). This necessitates knowledge about the relationship between the conceptual description and detailed description in a template or component, and knowledge related to practical implications of design choices.

The third distinction with previous work is that the client of the re-design process is not available for consultation during the re-design process, as it is the subject of the re-design process as well. The redesign process is based on the agent's self-awareness knowledge.

5.3 DISCUSSION

Automated design of adaptive software agents is a fascinating area of research. The interplay between client, artefact and designer plays an important role: adaptive artefacts are both client and subject of (re-)design. Artefacts are designed to be adaptive: they need to recognise needs for change. The agent factory adapts the agent according to their needs.

An agent needs self-awareness knowledge to determine needs for adaptation. The 'degree of' self-awareness of an agent determines the 'degree of' interpretative knowledge needed by an agent factory: the more imprecise an agent's needs for adaptation are, the more responsibility an agent factory has for interpretation of needs for adaptation.

This paper describes the relation between self-awareness knowledge and the knowledge available to an agent factory. Automated adaptation of agents (a re-design process) is not straightforward. Agents are designed at both a conceptual level and a detailed level: a configuration of conceptual building blocks and a configuration of detailed building blocks are needed.

Self-aware agents themselves decide which agent factory is to perform a

re-design task. Such self-modifying agents (Brazier and Wijngaards, 2001b) may evolve in ways that their designers could never have pre-conceived. This may be a testbed for creativity as both agents and agent factories have reflective capabilities, a pre-requisite for creativity.

The approach taken in the agent factory is similar, to some extent, to approaches such as IBROW (Motta, Fensel, Gaspari and Benjamins, 1999). In IBROW semi-automatic configuration is supported of intelligent problem solvers. Their building blocks are 'reusable components', which are not statically configured, but dynamically 'linked' together by modelling each building block as a CORBA object. The CORBA-object provides a wrapper for the actual implementation of a reusable component. A Unified Problem-solving Method development Language UPML (Fensel, Motta, Benjamins, Crubezy, Decker, Gaspari, Groenboom, Grosso, van Harmelen, Musen, Plaza, Schreiber, Studer and Wielinga, 2002) has been proposed for the conceptual modelling of their building blocks. The agent factory differs in a number of aspects, which include: multiple conceptual and detailed languages, no pre-defined wrappers for detailed building blocks, agents consist of one (multi-threaded) process, and the process of reconfiguration is an automated (re-)design process.

Current research within the IIDS group focuses on the design and implementation of an Agentscape: an agent operating system together with a set of services. The Agent Factory is one of the services. A number of prototype agent factories have been designed, implemented and tested, providing insight in the functionality required from an agent operating system, and the functionality to be provided to mobile agents aware of their options for adaptation.

Acknowledgements

The authors wish to thank the graduate students Hidde Boonstra, David Mobach, Oscar Scholten and Sander van Splunter for their explorative work on the application of an agent factory for an information retrieving agent. This work was supported by NLnet Foundation, `http://www.nlnet.nl/`.

References

Barber, K. S., McKay, R. M., MacMahon, M. T., Martin, C. E., Lam, D. N., Goel, A., Han, D. C. and Kim, J.: 2001, Sensible Agents: An Implemented Multi-Agent System and Testbed, in *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, New York: ACM Press, pp. 92-99.

Biggerstaff, T. J. and Perlis, A. J. (eds.): 1997, *Software Reusability. Volume 1, Concepts and*

models. New York: ACM Press.

Brazier, F. M. T., Jonker, C. M. and Treur, J.: 2000, Compositional Design and Reuse of a Generic Agent Model, *Applied Artificial Intelligence Journal*, **14**, 491-538.

Brazier, F. M. T., Jonker, C. M., Treur, J. and Wijngaards, N. J. E.: 2000, Deliberate Evolution in Multi-Agent Systems, *in* J. Gero (ed.), *Proceedings of the Sixth International Conference on AI in Design, AID'2000*, Dordrecht: Kluwer Academic Publishers, 2000, pp 633-650.

Brazier, F. M. T., Jonker, C. M., Treur, J. and Wijngaards, N. J. E.: 2001, Compositional Design of a Generic Design Agent, *Design Studies journal*, **22**, 439-471.

Brazier, F. M. T., Langen, P. H. G. van and Treur, J.: 1998, Strategic Knowledge in Compositional Design Models, *in* J. S. Gero and F. Sudweeks (eds), *Proceedings of the Fifth International Conference on Artificial Intelligence in Design, AID'98*, Dordrecht: Kluwer Academic Publishers, Dordrecht, pp. 129-147.

Brazier, F. M. T., Langen, P. H. G. van, Ruttkay, Zs. and Treur J: 1994, On formal specification of design tasks, *in* J. S. Gero and F. Sudweeks (eds), *Proceedings Artificial Intelligence in Design (AID'94)*, Dordrecht: Kluwer Academic Publishers, pp. 535-552.

Brazier, F. M. T., Splunter, S. van and Wijngaards, N. J. E.: 2001, Strategies for integrating multiple viewpoints and levels of detail, in J. S. Gero and K. Hori (eds), P*roceedings of workshop on Strategic Knowledge and Concept Formation III*, Sydney: Key Centre of Design Computing and Cognition, University of Sydney. December 2001, pp. 103-128.

Brazier, F. M. T. and Wijngaards, N. J. E.: 2001a, Automated servicing of agents, *Journal of AISB*, special issue on Agent Technology, **1**(1), 5-20.

Brazier, F. M. T. and Wijngaards, N. J. E.: 2001b, Designing Self-Modifying Agents, in J. S. Gero and M. L. Maher (eds), *Proceedings of Computational and Cognitive Models of Creative Design, the fifth international roundtable conference V*, Sydney: Key Centre of Design Computing and Cognition, University of Sydney. December 2001, pp. 93-112.

Broekstra, J., Kampman, A. and van Harmelen, F.: 2001, Sesame: An Architecture for Storing and Querying RDF Data and Schema Information, *in* D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (eds), *Semantics for the WWW*, Boston: MIT Press, pp. 272-309.

Chandrasekaran, B.: 1986, Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design, *IEEE Expert*, **1**(3), 23-30.

Fensel, D., Motta, E., Benjamins, V.R., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., van Harmelen, F., Musen, M., Plaza, E., Schreiber, A.Th., Studer, R. and Wielinga, B. J.: 2002, The Unified Problem-solving Method Development Language UPML, *Knowledge and Information Systems*, to appear.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: 1994, *Design Patterns: Elements of reusable object-oriented software*, Reading, Massachusetts: Addison Wesley Longman.

Gray, R., Kotz, D., Cybenko, G. E. and Rus, D.: 1997, Agent Tcl, *in* W. Cockayne and M. Zyda (eds), *Mobile Agents: Explanations and Examples*, Greenwich, CT: Manning Publishing, 1997, pp. 58-95.

Grefenstette, J.: 1992, The evolution of strategies for multiagent environments, *Adaptive Behavior*, **1**(1), 65-90.

Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D. and Stein, L. A.: 2001. DAML+OIL, http://www.daml.org/2001/03/daml+oil-index.html.

Kolodner, J. L.: 1993, *Case-Based Reasoning*. San Mateo, California: Morgan Kauffman.

Maher, M. L. and Pu, P. (eds): 1997, *Issues and Applications of Case-Based Reasoning to*

Design, Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Motta, E., Fensel, D., Gaspari, M. and Benjamins, V.R.: 1999, Specifications of Knowledge Component Reuse, *in Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE-99)*, Skokie, IL: Knowledge Systems Institute, pp. 17-19.

Mozer, M. C.: 1999, An intelligent environment must be adaptive, *IEEE Intelligent Systems and their Applications*, **14**(2), 11-13.

Nwana, H., Ndumu, D., Lyndon, L., and Collis, J.: 1999, ZEUS: A Tookit and Approach for Building Distributed Multi-agent System, *in Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, New York: ACM Press, pp. 360-361.

Peña-Mora, F. and Vadhavkar, S.: 1996, Design Rationale and Design Patterns in Reusable Software Design, *in* J. S. Gero and F. Sudweeks (eds.), *Artificial Intelligence in Design (AID'96)*, Dordrecht: Kluwer Academic Publishers, pp. 251-268.

Reffat, R. M. and Gero, J. S.: 2000, Computational Situated Learning in Design, in: J. S. Gero (ed), *Artificial Intelligence in Design '00*, Dordrecht: Kluwer Academic Publishers, Dordrecht, pp. 589-610.

Reticular Systems Inc: 1999, AgentBuilder: An integrated toolkit for constructing intelligent software agents. *White Paper*, http://www.agentbuilder.com, February 1999.

Rus, D., Gray, R. and Kotz, D.: 1996, Autonomous and Adaptive Agents that Gather Information, *in AAAI'96 International Workshop on Intelligent Adaptive Agents*, AAAI Technical Report WS-96-04, pp. 107-116.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W. and Wielinga, B.: 1999, *Knowledge Engineering and Management, the CommonKADS Methodology*, Cambridge, MA: MIT press.

Schreiber, A. Th. and Birmingham, W. P. (eds.): 1996, Special Issue on Sisyphus-VT. *International Journal of Human-Computer Studies (IJHCS)*, **44**(3/4), 275-280.

Sparling, M.: 2000, Lessons learned through six years of component-based development. *Communications of the ACM*, **43**(10), 47-53.

Stefik, M.: 1995, *Introduction to Knowledge Systems*, San Francisco, California: Morgan Kaufmann Publishers, Inc.

Stumptner, M. and Wotawa, F.: 1998, Model-Based Reconfiguration, *in* J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design, AID'98*, Dordrecht: Kluwer Academic Publishers, pp. 45-64.

Suri, N., Bradshaw, J. M., Breedy, M. R., Groth, P. T., Hill, G. A., Jeffers, R. Mitrovich, T. S., Pouliot, B. R. and Smith, D. S.: 2000, NOMADS: Toward a Strong and Safe Mobile Agent System, *in Proceedings of the Fourth International Conference on Autonomous Agents*, New York: ACM Press, pp. 163-164,

Tryllian: 2001, Agent Development Kit, *Technical White Paper*, Version 1.0, June 2001, http://www.tryllian.nl/sub_downl/Technical%20white%20paper%20ADK%20v1.0.pdf

Wells, N. and Wolfers, J.: 2000, Finance with a Personalized Touch, *Communications of the ACM*, Special Issue on Personalization, **43**(8), 31-34.

Williams, B. C. and Nayak, P. P.: 1996, A Model-Based Approach to Reactive Self-Configuring Systems, *in Proceedings of the AAAI'96 & IAAI'96*, Cambridge, MA: AAAI Press / MIT Press, **2**, pp. 971-978.